THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 9, November 1982
Printed in U.S.A.

Database Systems:

Implementation of a Distributed Database Management System to Support Logical Subnetworks

By D. COHEN

(Manuscript received November 3, 1981)

This paper describes the design and implementation of a special-purpose distributed data management system. The design and implementation were parts of a study to evaluate the database management needs of software-defined logical subnetworks. The paper describes the authorization model used to define logical subnetworks and the related subnetwork management transactions. The definition of the subnetworks and their service capabilities are stored in a distributed database. The distributed database architecture and the implemented software architecture are described. The requirement to design and implement within a specific time frame has kept the design simple, but the nature of the application dictated that we consider many aspects of the more general distributed data-management problem. The database management issues that are addressed in this paper, in the context of transaction processing, include multicopy updates, concurrency control, and crash recovery. A version of the primary node concept for multicopy updates was adopted. Data inconsistencies, created by premature termination of transaction processing (e.g., system crash), are detected and removed by the software.

I. INTRODUCTION

The advantages of distributed data processing in general, and distributed data management in particular, have been presented in many publications. In spite of wide interest in its potential cost benefits,

distributed data management has met little acceptance in the field because of its potential impact on the way organizations are accustomed to managing their data-processing facilities.[1,2] Issues such as who is responsible for purchasing of equipment, or who is responsible for availability of services have to be revisited in the distributed processing environment. This paper presents the design of a distributed database management system. The implementation was part of a special study investigating the feasibility of software-defined logical subnetworks. However, the design is general in that many issues of distributed database management are addressed and solved.

The database contains the definition of logical subnetworks, their users, and the service features to which the users subscribe. In software-defined subnetworks, customers can be provided direct control over their own subnetwork. This aspect of the service is referred to as customer subnetwork management. The customer subnetwork capabilities deny customers control over other customers' subnetworks, and protect the network from customer-initiated activities. The database architecture selected, in the context of a nation-wide service, can be applied to a variety of communication services.

Section II defines the requirements placed on the database management system by the application. Section III describes the database architecture selected in response to the reliability and performance requirements of the service. Section IV describes the software architecture developed to maintain the distributed databases. Section V presents solutions to major database issues.

II. APPLICATION

The specific application under study was in the context of a communication service—a service assumed to be operational 24 hours a day, 7 days per week.

2.1 Definitions

2.1.1 The network

The service is handled by a collection of physical nodes. Each physical node supports *one or more* logical nodes, called *service areas* (SAs). A service area cannot span physical nodes. A service area provides service through a set of network addresses. An SA is identified by a six-digit number. A network address is identified by a ten-digit number that includes the six digits of the associated SA. For example, 201-777 is a service area name, and 201-777-4444 is a network address associated with it. Figure 1 shows the network as a collection of physical nodes, each supporting one or more service areas.

2.1.2 Users

A user is an entity that can be provided service. Users are grouped

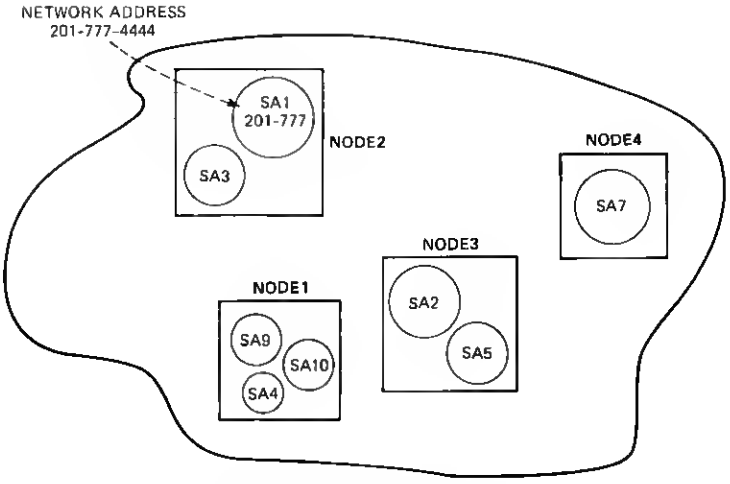

Fig. 1—Nodes and service areas in the network.

according to their service capabilities (e.g., set up calls, terminate calls, disable service to another user, etc.). At any one time, a user may only be a member of a single *group*. All the users of a given group share the same set of capabilities. A *customer account* is a collection of user groups. A user is identified by the combination of a *group name* and a *home address*. The home address of the user is a ten-digit number and it identifies the service area that is normally used by him/her to access service. But, a user may access (if authorized) network services through network addresses associated with any other service area. The customer account name is part of the group name. For example, BELL, BELL.RESEARCH, BELL.RESEARCH:201-582-9999 are account, group, and user names, respectively. An instance of a user requesting service at a particular address is referred to as an *active user*. Users of an account are provided service by their home SAs, which were selected by the customer. Therefore, an account may span several SAs according to the distributed nature of the customer's business.

Each user is assigned a data profile. Actually, a user's data profile is constructed from his/her account, group, and home address profiles. A user profile contains a list of the service features subscribed to by the user and some status information. These may include features to transfer information between users (communication features such as "call setup" and "send message") and/or features to control other users' service capabilities (subnetwork management features such as "disable and account," "a group," or another user).

2.2 The authorization model

A shared network must preserve the rights of its users. The author-

ization model defines the format of the *authorization policy* specified by the customer, and the *enforcement mechanism* used to control user access to the service.

2.2.1 The authorization policy

The collection of users, network addresses, and groups associated with an account is referred to as a *logical subnetwork*. The authorization policy of an account is the database representation of its logical subnetwork. Logical subnetworks can be created or removed only through service provisioning, i.e., changes in the database. The capability of a user to modify the authorization policy of an account is referred to as *subnetwork management*. The operations support users are also organized in one or more accounts. These accounts are identified through reserved names and associated users may be assigned to customers. Subnetwork management is used by operations support users to create, modify, and remove logical subnetworks. Customers are provided access to some limited customer subnetwork management capability. Users who can exercise customer network management capabilities are referred to as administrators. Administrators are restricted in their scope of control to their own customer subnetworks. For example, an administrator in one account cannot disable service to a user from another account. An administrator can modify the authorization data associated with existing users, but he/she cannot create or remove either a user or a network address.

We recall that an account consists of one or more groups. The implementation supports account and group administrators. An account administrator can exercise control over his/her whole account. For example, the account administrator can disable service to his/her whole account, to a group, or to a specific user. A group administrator is assigned control over one or more groups within his/her own account, but a group can be assigned only to a single group administrator. Group administrators can exercise subnetwork management functions only over groups under their control. Group administrators, by definition, cannot be members of a group under their control. The subnetwork management capabilities available to customer administrators differ only in their scope of control. For example, an account administrator of account A1 has the same capabilities as account administrator of account A2. But each administrator is restricted to exercise his/her capabilities within his/her own account.

In summary, the authorization policy of an account is stored in a set of data profiles associated with logical subnetwork entities such as users, network addresses, and groups.

2.2.2 The enforcement mechanism

The authorization policy is used by the enforcement mechanism to

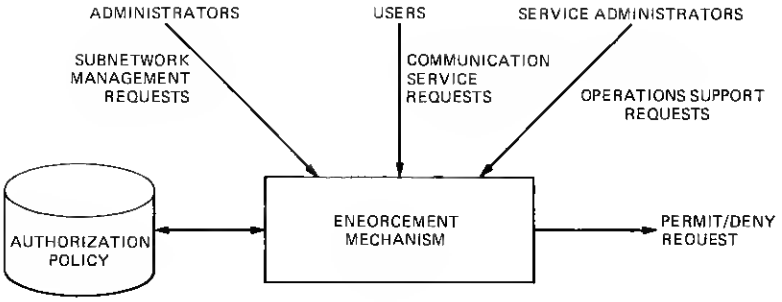

Fig. 2—The enforcement mechanism.

authorize all user service requests. This applies uniformly to requests issued by customer or operations support users. With respect to customers, the enforcement mechanism is invoked to authorize both communication and subnetwork management service requests. Figure 2 illustrates the enforcement mechanism and the related authorization policy as defined by the model. A customer service request is authorized using the policy stored in the related user profile. To minimize the enforcement delay associated with a service request, a copy of the user profile is cached in main memory at the service area used by the active user to access the network. The following section describes the authorization database architecture used to maintain software defined subnetworks.

III. DATABASE ARCHITECTURE

3.1 Is distributed database management necessary?

The application described in Section II specifies a use of data management to support the maintenance of user profiles. The need for distributed database management was one of the most important design decisions. An evaluation of the application showed that distributed database management is necessary for four major reasons: load sharing between the nodes, enforcement mechanism performance, communication service reliability, and node software standardization.

User profiles must be available at the node which provides service to support authorization enforcement. If all user profiles were stored at a single node, the system could not meet its performance and availability objectives. The node that maintains all user profiles may become a bottleneck whenever large numbers of users wish to initiate service concurrently. It was also concluded that it would be disadvantageous to develop and maintain two different node software versions, one for the node maintaining the profiles, and another for the nodes providing communication services. Therefore, it was decided to dis-

tribute the database management load associated with user profiles across all service areas (all nodes).

3.2 Physical distribution of data

User profile data is distributed at three different levels to minimize enforcement delay associated with communication service requests, to improve availability of the service, and to maintain consistency of replicated data elements. At any point in time, a user's profile may exist at: the account's SA (first level), the user's home SA (second level), and the access SAs that support all related active users (third level). Data at the first level is used to coordinate the update of replicated data elements that have been stored at all three levels. Data at the second level is used to enhance the availability of service. As long as the user's home SA is operational, a user may be guaranteed service by this site. Data at the third level is stored in volatile memory and is used mainly to decrease the delay associated with enforcement.

3.2.1 The first level

All authorization data associated with an account (e.g., group profiles, user profiles, address profiles, etc.) are stored in a *customer authorization database* (CADB) at a specific service area. This service area is referred to as the account's SA (or the first level). The CADB of a service area may contain the authorization data of one or more accounts. The authorization model does not allow any logical data dependencies between accounts. This design decision simplifies significantly the maintenance of data integrity of logically related profiles within an account. For example, the system rejects a "create user" transaction if the related group and home address profiles do not exist. The account's SA is selected by operations support personnel according to load-balancing considerations, independently of the service areas that will provide service to users.

The primary node concept for multicopy updates was chosen to support profile updates.[3-6] All changes in authorization information will occur first in the CADB, and will be propagated to the other levels if necessary.

3.2.2 The second level

Whenever a user is created, a user profile is downloaded from the CADB to the users' home SA (or the second level), and stored in nonvolatile memory. All secondary copies at a service area are stored in the *secondary copy database* (SCDB). A SCDB includes profiles of all users homing on this service area, independently of their account membership. Users can be provided service as long as their home SA is operational, even if their account's SA is not.

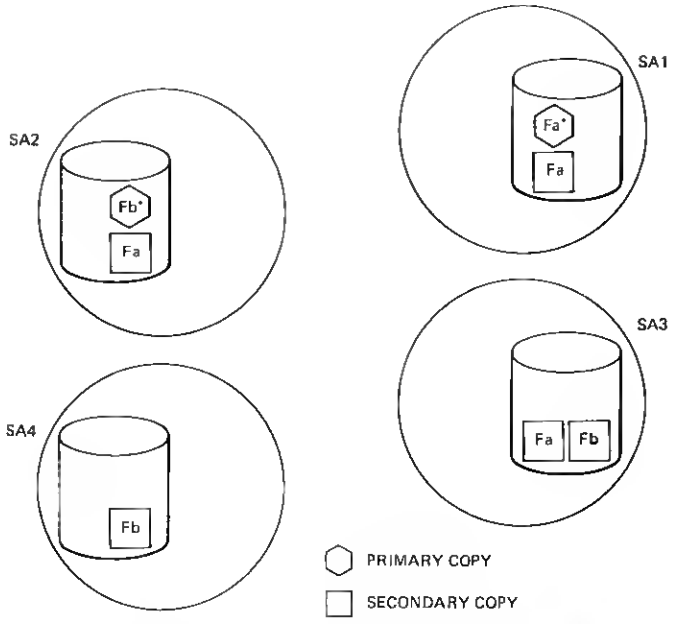

Fig. 3—Primary copy concept—multiple copy updates.

Secondary profile copies can be updated only through the account's CADB. Therefore, subnetwork management requires that both the user's home SA and the account's SA be operational. Figure 3 shows an instance of the primary copy concept. Profile Fa has its primary copy installed at SA1, SA2, and SA3. Similarly for profile Fb, the primary copy is installed at SA2 (Fb'), and secondary copies exist at SA3 and SA4. A secondary copy of profile Fa is installed at SA1 because the primary copy *is* not used to support enforcement. This decision simplifies the development of the software responsible for migrating CADBs and SCDBs, independently from one node to another. For example, the CADBs of several accounts can be homed to a new service area at a different node without any changes in the content of the SCDBs at the old node. Network growth and/or load balancing considerations will require the migration of databases between nodes.

3.2.3 The third level

A user may be provided service through a network address only after a log-on procedure verifies his/her identitiy. If log-on is successful, a session associated with an active user is established. During a session, an active user may submit one or more service requests. For each active user, a copy of his/her profile is cached in main memory at the access service area (or the third level). This may or may not be the

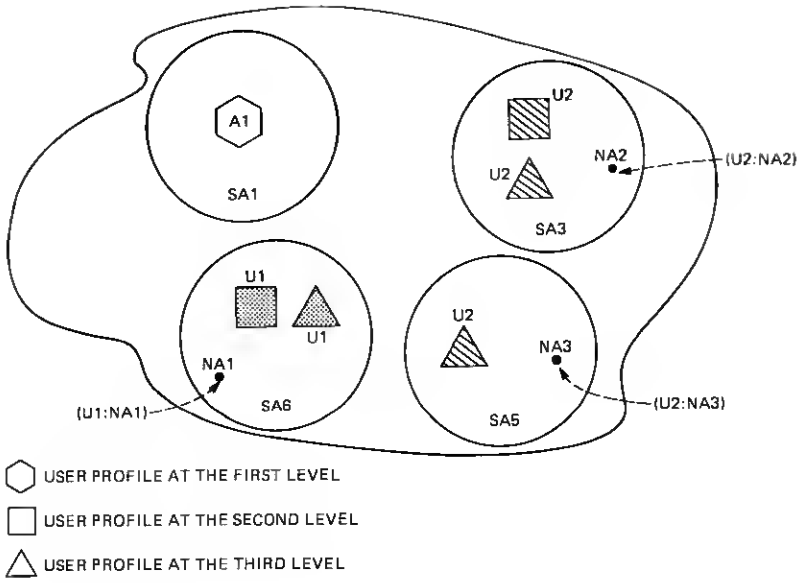

Fig. 4—User profiles.

home service area of the user. The cached profile copy is created to further minimize enforcement delay associated with communication service requests. The active user profile is kept in volatile memory for the duration of the session. The system can provide service concurrently to one or more active instances of a user. The user's home SA keeps track of all active instances of a user. Figure 4 shows account A1 homing on service area SA1. Two users of this account, U1 and U2, home on service area SA6 and SA3, respectively. An active instance of user U2 is created when he/she requests service through network address NA3, associated with service area SA5. Other active users, U1:NA1 and U2:NA2, are provided access through service areas SA6 and SA3, respectively.

Each node maintains a routing table which identifies the SAs of all accounts. Update transactions are routed, using this table, to the appropriate account's CADB. Each profile in the CADB maintains a list of SAs, where related secondary copies have been installed. This information is used to propagate update transactions to the second level.

Changes to a profile at the third level take effect after session termination of the affected active user. All new sessions are established using the updated version of the profile at the second level. To minimize interference during a session, the administrator may or may not request immediate activation of the change. If an immediate option

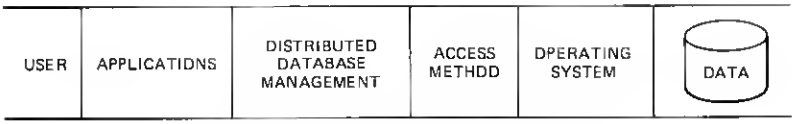

Fig. 5—Functional partitioning of the software.

is specified, sessions of all affected active users are terminated. If the immediate option is not specified, the update at the second level will affect only future sessions.

In the rest of this paper, discussions will be restricted to issues associated with the first and second levels of the database architecture.

IV. SOFTWARE ARCHITECTURE

The *UNIX** operating system was chosen as the implementation environment.[7] At the time of the study, a commercial database management system controlled by the *UNIX* operating system was not available. To limit development resources, a decision was made to concentrate most of the effort on the distributed aspects of the problem. The centralized database management system supports file directories, sequential files, and single key files, built over the *UNIX* file system. The assumption was that centralized database management could be upgraded in the future, once a commercial package became available. Figure 5 shows the functional partitioning of the software. The design effort concentrated on key issues such as multicopy updates, concurrency control, and crash recovery. Tight dependencies were detected between these issues. Similar dependencies were reported in Ref. 6. Section V presents some of these dependencies and their impact on implementation.

4.1 Process structure

The two functions associated with multicopy updates of the CADB and SCDB are supported by two data managers. The first manager, referred to as the *primary copy data manager* (PCDM), updates profiles at the first level and coordinates the distribution of secondary copy updates to the affected SCDBs. The second manager, referred to as the *secondary copy data manager* (SCDM), updates profiles at the second level—stored in the SCDB—in response to PCDM requests and supports enforcement at the user's home service area. The two data managers are implemented as separate processes because of size limitation imposed by the operating system. Figure 6 shows the uniform process structure selected for all service areas.

* Trademark of Bell Laboratories.

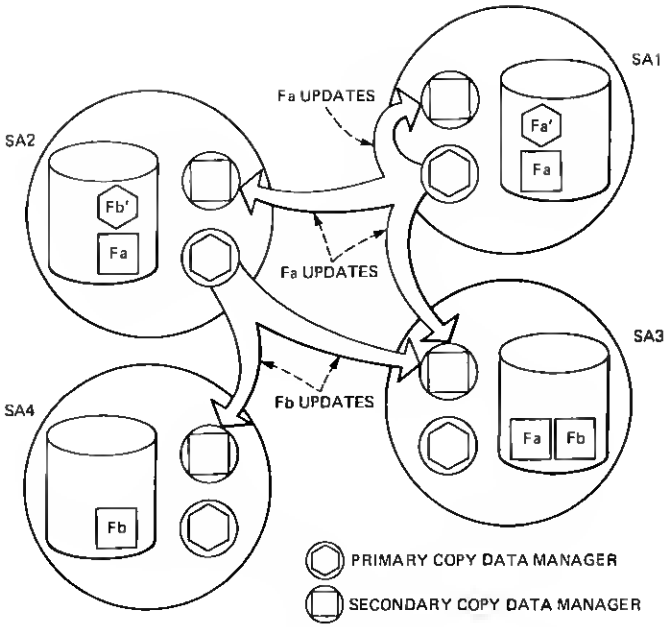

Fig. 6—Data management process structure.

The design of both data managers follows a layered approach with well-defined interfaces. This facilitated parallel development of the layers, and allows easy upgrading of existing layers in the future. Database updates of first-level copies are supported through four layers. The first layer provides local data access and secondary storage management. The second layer (PCDM) supports centralized control of updates, data consistency within the primary copy's node, concurrency in transaction processing, deadlock prevention, and crash recovery. The third layer allows data distribution to be transparent to node data managers. The fourth layer, as a user interface, transforms user queries into database transactions. This layer also includes an access control mechanism based on the capability model.[8,9]

Enforcement requests and updates of secondary copies are supported through two layers. The first layer provides local data access (the same one as for the first-level copies). The second (SCDM) updates secondary copies and supports enforcement. Figure 7 shows the database management system's layered architecture.

V. TRANSACTION PROCESSING

The design process was implementation driven. In all cases in which a simple solution was available, it was adopted. Initially, we decided to review issues one at a time. However, we soon discovered that some

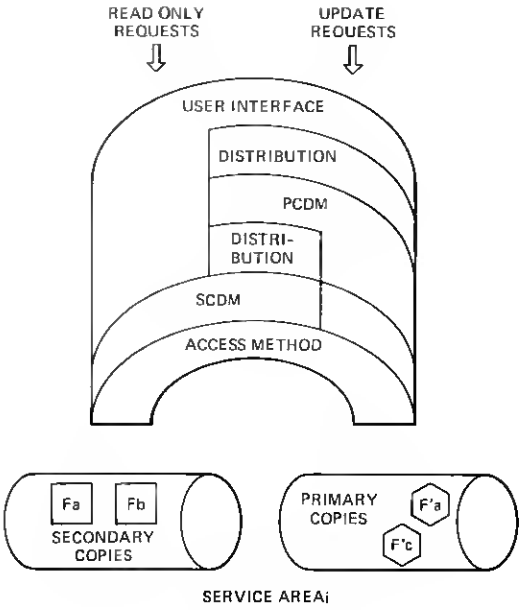

Fig. 7—System software architecture.

aspects of distributed data management are interrelated and cannot be examined separately. For example, the update algorithm had to be revised numerous times while crash recovery was investigated. As long as crash recovery was not considered, all update algorithms were satisfactory. The following sections review the transaction types supported, the update algorithm, and the crash-recovery mechanism.

5.1 Input transaction types

The PCDM accepts three types of transactions. (*i*) *Customer subnetwork transactions* are issued by an administrator to exercise control over his/her logical subnetwork. These include access requests to either verify or modify the status of the logical subnetwork. (*ii*) *Process control transactions* are issued by maintenance personnel to monitor and control PCDM functionality. For example, these transactions can be used to disable the processing of a particular type of customer subnetwork transaction for all customers because of a software problem. (*iii*) *Response transactions* are issued by SDCMs in response to PCDM-initiated update transactions.

Process control and response transactions have a uniform high priority, and are processed before all new subnetwork management transactions. Customer subnetwork transactions are processed by the PCDM at one of three priorities. Administrators may specify the priority

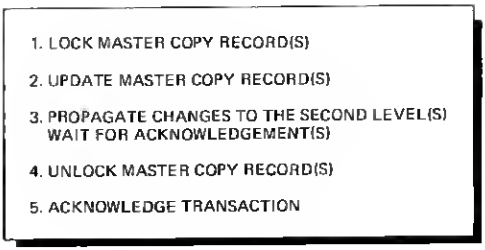

Fig. 8—Update algorithm.

of each transaction submitted. If not specified, a default priority is assigned by the system uniformly for all customers.

The SCDM accepts three types of transactions. (*i*) *Update transactions* are issued by PCDMs to create, remove, or modify profiles in the SCDB. (*ii*) *Process control transactions* are issued by maintenance personnel to monitor and control SCDM functionally. (*iii*) *Enforcement transactions* are issued by other processes to verify the capabilities of a user in the context of a service request. A user's request to initiate a session, or to make a call to another user are examples of requests that have to be authorized by the SCDM.

Process control and enforcement transactions have a uniform high priority, and are processed before all new update transactions. All update transactions are processed at a single-priority level because they are issued by the PCDM.

5.2 The update algorithm

The update algorithm handles transactions to add, modify and remove the profile. Each update transaction is routed to the PCDM at the account's home service area. The PCDM updates the first-level copy(ies) in the CADB and coordinates the update of all related secondary copies if necessary. Once all secondary copies have been updated, the PCDM returns an acknowledgment to the user interface process which originated the transaction. Figure 8 describes the update algorithm. The algorithm is similar to phase one of the two-phase commit protocol.[2] But, once the transaction is received at the SCDM, the response to the PCDM represents the fact that the change has been committed. If one or more SCDMs do not acknowledge completion, the PCDM terminates the transaction and notifies node maintenance of the potential existence of a database inconsistency. Node maintenance is provided with software tools to restore database consistency. The role of these tools is further discussed as part of the recovery strategy.

5.2.1 Locking

The locking mechanism supported by the system has four major

functions. First, it is used for concurrency control to enforce the serializability of multisource updates.[10,11] For example, different active users may submit authorized update transactions related to the same profile. The locking information is maintained in a system lock table and not with the data entities being updated.

Second, it is used to recover update transactions that have been abnormally terminated as a result of a system failure. The lock table is maintained in nonvolatile memory. If the lock table survived the crash, it is used by the recovery process to identify potential database inconsistencies. The lock table contains sufficient information to restore database consistency using roll forward or backward techniques.[12]

Third, it is used to prevent data-resource-related deadlocks.[13] Each update transaction locks a priori all data elements that are to be modified at the CADB, in the first step of the update algorithm. Most updates require the locking of two or three records in the CADB. The system supports locking granularity at the database, the file, and the record level. Using the primary copy concept, locking at the first level also implies locking at the other levels. Therefore, it is not necessary to acquire and release data locks across different nodes. If a transaction cannot acquire all the locks it needs, the system suspends its processing, and an attempt is made by the system to retry it later.

Fourth, the locking mechanism is used to enforce the CADB order of updates at the respective SCDBS. The communication network used to propagate changes from PDCMS to SCDMS does not ensure the delivery of transactions in the order they were sent out.[14] To prevent "race" conditions, the update algorithm releases the locks only after all secondary copies have been updated.

To prevent locking of data for long periods of time, the PCDM times-out each update transaction propagated to an SCDM. If acknowledgment is not received within a predefined time interval, the transaction is aborted and node maintenance is notified. Retransmitting update transactions to nonresponding SCDMS was not found useful. The loss of the initial update transaction is the only case where a retransmission is useful, but this is a rare occurrence in existing communication networks. Whenever the SCDM is overloaded or out of service, retransmissions may actually worsen the situation.

5.3 Crash recovery

The authorization policy is stored at three levels. Failures which disable access to these databases may result in communication and/or subnetwork management service interruption. This makes the recovery mechanism a critical element of the database management system. The main objectives of the recovery mechanism are to restore service by a service area as soon as possible, and to minimize the permanent

loss of data. It is also important to eliminate recovery dependencies between nodes of the network. A node should be able to recover on command, by itself, and restore service. The design assumes that database inconsistencies will occur, but service does not depend on complete consistency. The software is designed to detect database inconsistencies during normal transaction processing. Once an inconsistency is detected, node maintenance is notified. Node maintenance is provided a set of synchronization transactions to restore consistency within the first level, or between the first and second levels. These transactions are similar to regular update transactions, and differ only in the way they handle error conditions. Only after this manual method proves itself in the field will we consider automatic removal of database inconsistencies.

Appendix A provides a summary of the components used in recovery. Appendix B describes the PCDM and SCDM recovery algorithms used to restore service.

VI. SUMMARY

The development of a distributed database management system and of a subnetwork management application has been completed. The study demonstrated the feasibility of software defined subnetworks associated with a single nationwide network. We presented in this paper distributed database techniques and the feasibility of their implementation. These integrated database techniques are currently applied in the design of new communication services.

One of the issues not addressed in this paper is the development of tools for testing and debugging in a distributed processing system. The effort required to develop these tools was equal in scope to the development of both the PCDM and SCDM. The results in this area will be reported separately.

VII. ACKNOWLEDGMENTS

I wish to thank D. F. Hayden and M. M. Rochkind for their support and encouragement during this project; E. M. Rifkin for his assistance in the design and implementation of the data managers; J. A. Santillo for her implementation of the local database capability; J. E. Massery, F. K-F. Ng, R. G. Kayel, and M. B. Sury for their suggestions which helped improve the clarity of this manuscript.

REFERENCES

1. G. M Booth, "Distributed Data Bases, Their Structure and Use," INFOTECH State of the Art Report: Distributed Systems, 1976, pp. 201–13.
2. J. B. Rothnie and N. Goodman, "A Survey of Research and Development in Distributed Database Management," Proc. Third Int. Conf. on Very Large Data Bases, Tokyo, Japan, 1977, pp. 48–62.

3. R. H. Thomas, "A Solution to the Update Problem for Multiple Copy Databases Which Uses Distributed Control," Bolt Beranek and Newman Report No. 3340, July 1975.
4. P. A. Alsberg and J. D. Day, "A Principle for Resilient Sharing of Distributed Resources," Report from the Center for Advanced Computation, University of Illinois, 1976.
5. C. A. Ellis, "Consistency and Correctness of Duplicate Copy Database Systems," Proc. 6th ACM Symp. on Operating Systems Principles, November 1977, pp. 67–84.
6. M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," Third Berkeley Workshop on Distributed Data Management and Computer Networks, 1978, pp. 235–58.
7. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J, *57*, No. 6 (July–August 1978), pp. 1905–29.
8. G. S. Graham and P. J. Denning, "Protection Principles and Practice," Proc. Spring Joint Computer Conf., *40*, 1972, pp. 417–29.
9. A. K. Jones, "Protection in Programmed Systems," Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, June 1973.
10. P. A. Bernstein et al., "Analysis of Serializability in SDD-1: A System for Distributed Databases," Computer Corporation of America, *Report* No. CCA-77-05, 1977.
11. P. A. Bernstein, J. B. Rothnie, D. W. Shipman, and N. Goodman, "The SDD-1 Redundant Update Algorithm," Computer Corporation of America, *Report* No. CCA-77-09, 1977.
12. J. N. Gray, "Notes on database operating systems," IBM Research Report, RJ-2188, February 1978.
13. E. A. Menasce and R. R. Muntz, "Locking and Deadlock Detection in Distributed Data Bases," IEEE Trans. on Software Engineering, *SE-5*, No. 3 (May 1979), pp. 195-202.
14. F. Heart et al., "The Interface Message Processor for the ARPA Computer Network," SJCC AFIPS Conf. Proc., *36*, 1970, pp. 551–87.

APPENDIX A

Components of the Recovery Mechanism

Each node is equipped with the following components to be used in the recovery process:

(*i*) Duplex hardware is provided to protect service against single hardware failures.

(*ii*) The operating system maintains two copies of each data element on separate disk drives.

(*iii*) The node creates periodically a local, off-line, backup copy of the database for checkpointing.[12]

(*iv*) During transaction processing, the system maintains on disk the lock table and a completed update transaction log. The completed update transaction log is maintained at the physical page level and is used only by the recovery process. Physical logging was selected to improve performance during crash recovery when the database is rolled forward. The log maintains sufficient information to roll the database both forward and backward.[12] The log is kept until the next checkpoint is established. Audit trails for subnetwork management transactions are maintained separately at the logical level by the application programs.

(*v*) Node maintenance has a set of synchronization transactions for restoring database consistency at all levels.

APPENDIX B

The Recovery Algorithm

Using the recovery components specified in Appendix A, service can be restored in the following way:

(*i*) At the PCDM
   a. Check file consistency on disk.
   b. If both disk copies are lost, install backup copy, and internally run the completed update transaction log. Interactions with other processes, as part of normal transaction processing, are not necessary.
   c. Use the lock table to back out prematurely terminated transactions (synchronize within the first-level database and between the first-level and second-level databases).
   d. Renew subnetwork management service.
   e. Detect inconsistencies during normal transaction processing and notify node maintenance.

(*ii*) At the SCDM
   a. Check file consistency on disk.
   b. If both copies of the database on disk are lost, install backup, and internally run the completed update transaction log. Interactions with other processes, as part of normal transaction processing, are not necessary.
   c. Renew communication service.
   d. Inconsistencies detected during normal transaction processing are forwarded to node maintenance.